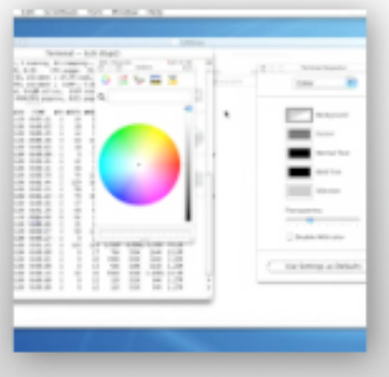
vImage

Apple's Vector-accelerated Image Processing Library

Session 103



vImage

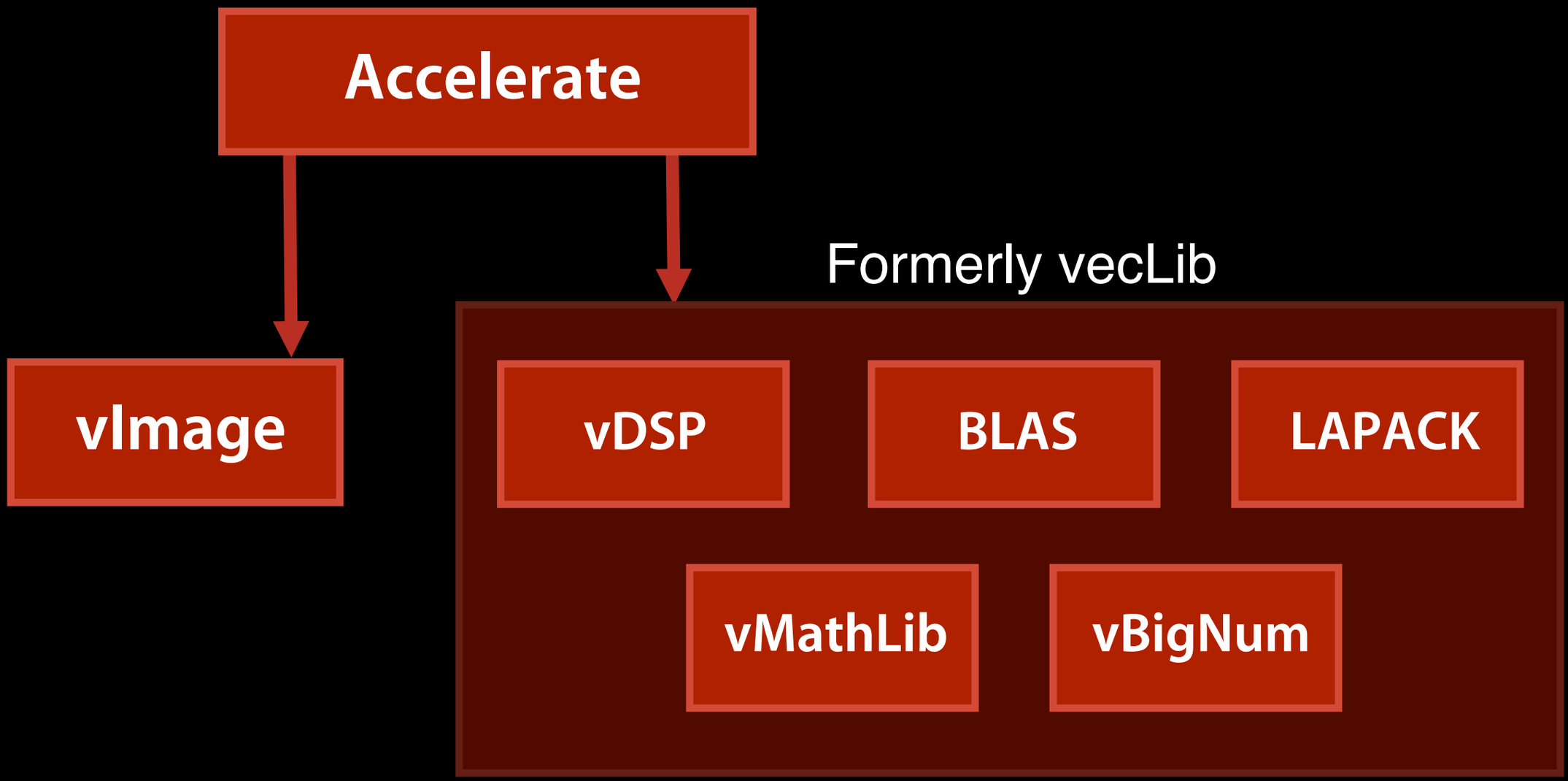
Apple's Vector-accelerated Image Processing Library

Craig Keithley
I/O Technology Evangelist
keithley@apple.com

Accelerate:
Apple's State of the Art, High Performance Computing Framework

Robert Murley
Sr. Engineering Scientist
Vector and Numerics Group

Technology Framework

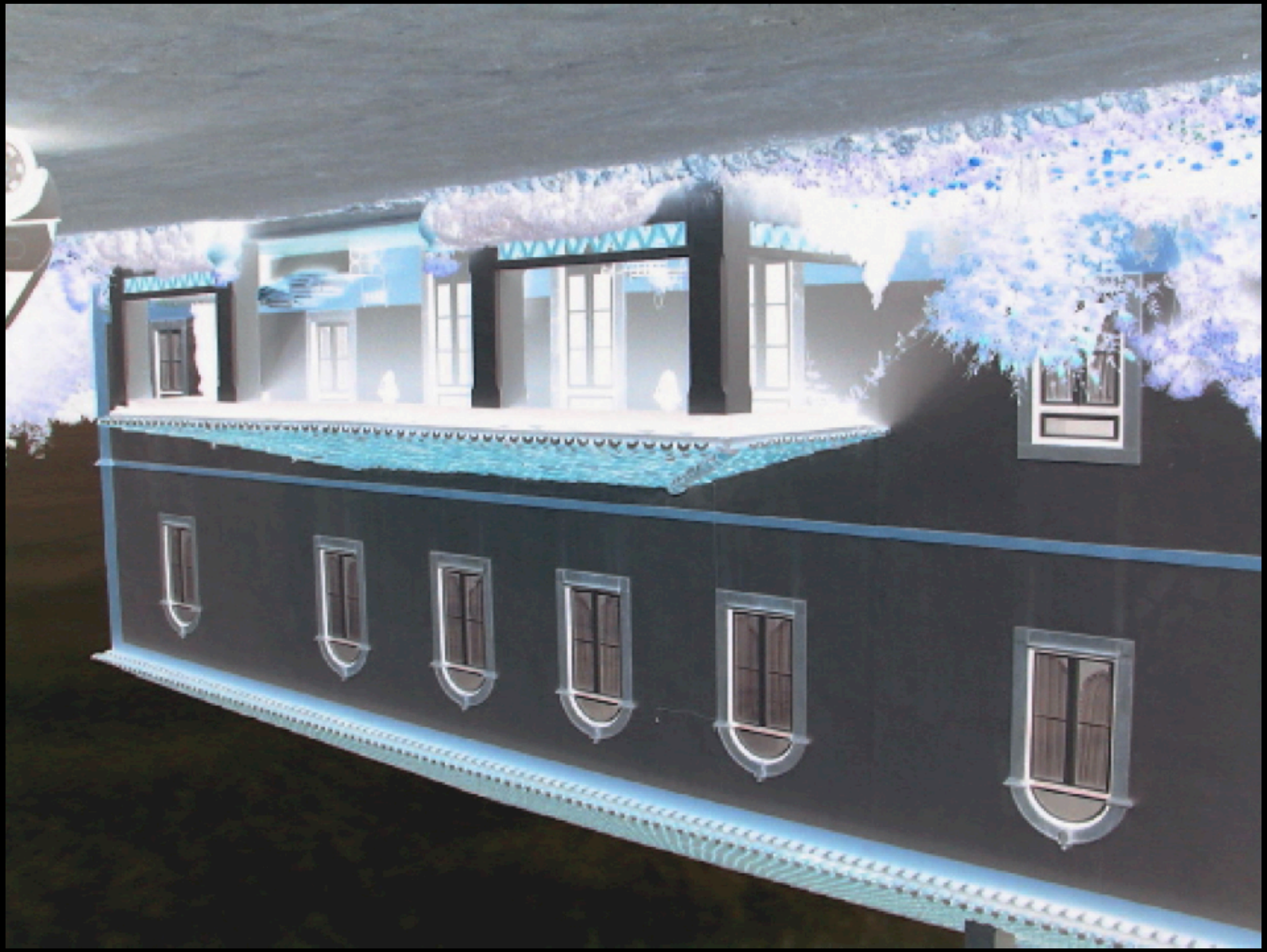


What You'll Learn About vImage

- Functionality
- Data structures and API
- Features
- Implementation Techniques
- Performance

Functionality

Convolution
Morphology
Geometry
Histogram
Alpha compositing
Data type conversion

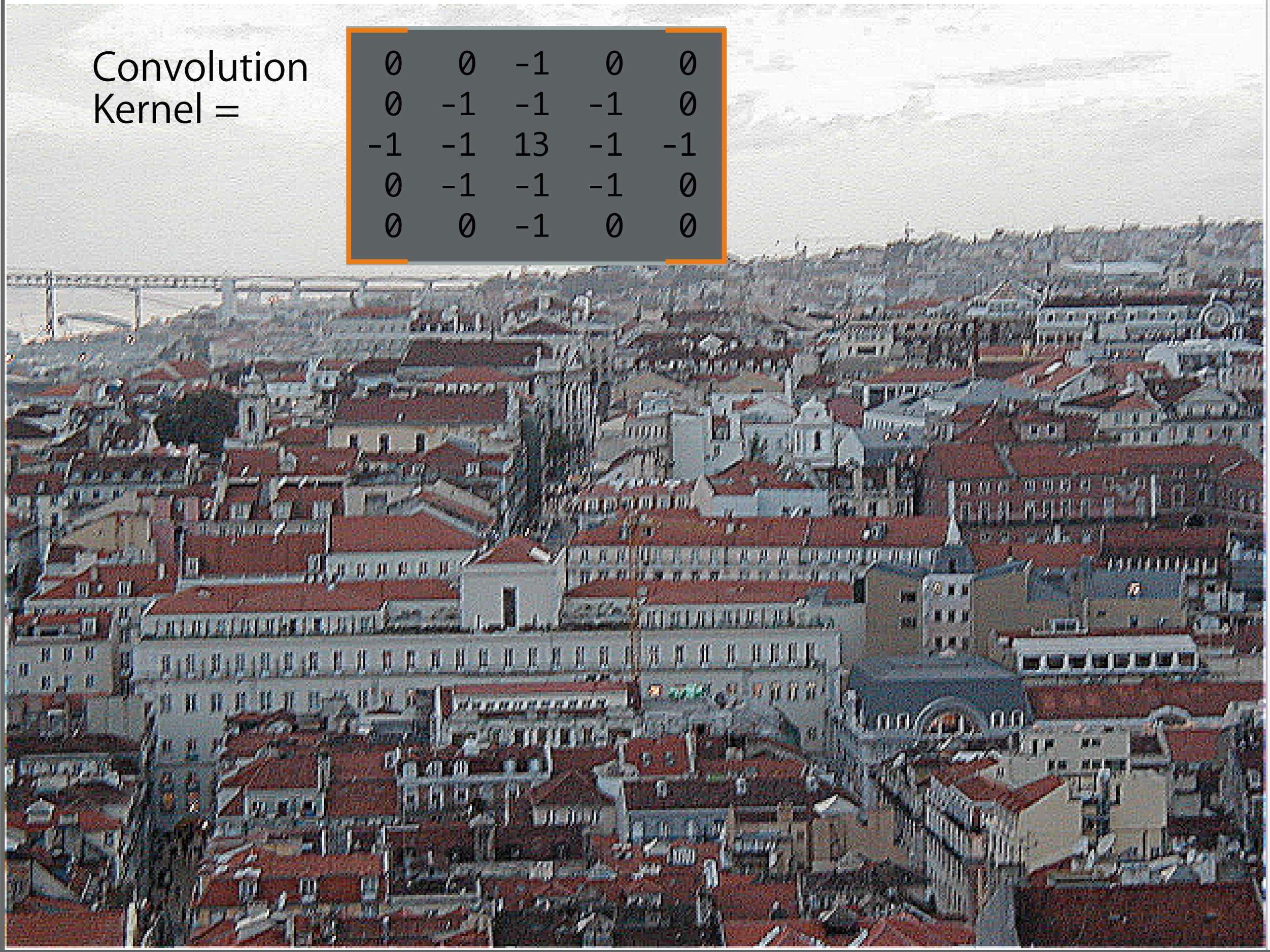


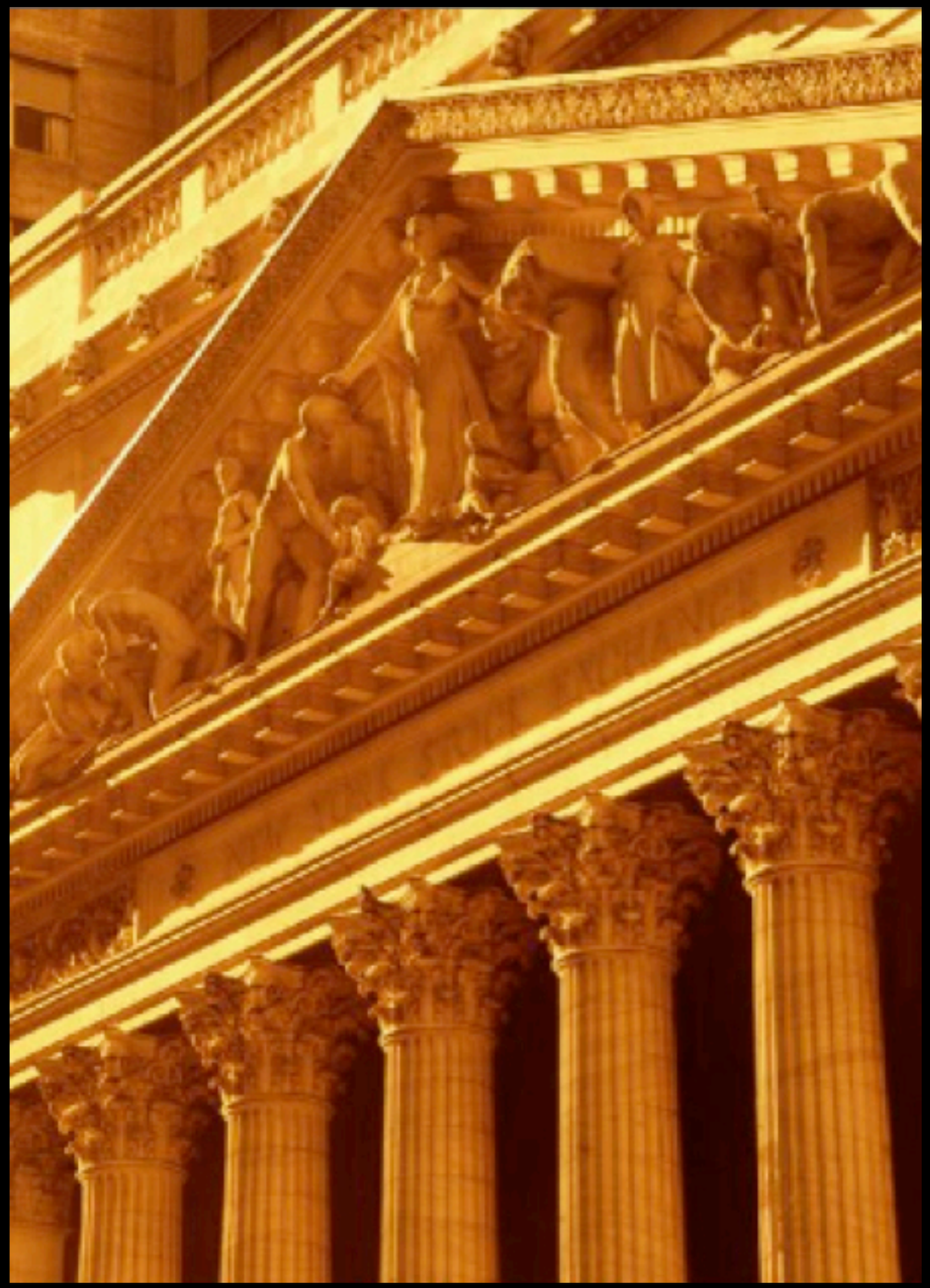
Convolution

- Blur/Sharpen
- Emboss
- Average
- Edge-Detection
- Thresholding
- Derivatives
- Gradients

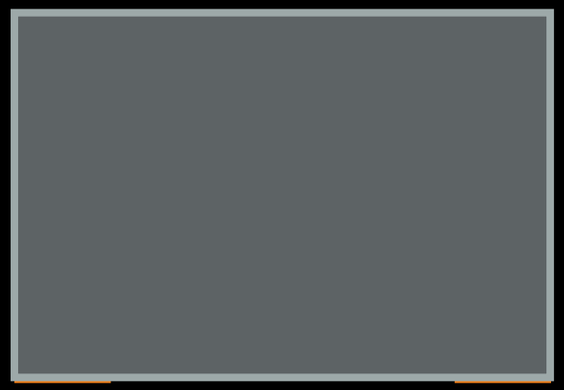
Convolution Kernel =

$$\begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -1 & -1 & 0 \\ -1 & -1 & 13 & -1 & -1 \\ 0 & -1 & -1 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix}$$





Convolution
Kernel =



Morphology

- Min/Max
- Dilate/Erode

Morphology Filters

Starting Image: Small and Large Circle

Dilate Filter

Morphology
Kernel =

$$\begin{bmatrix} 0 & 255 & 255 & 255 & 255 & 255 & 255 \\ 0 & 0 & 255 & 255 & 255 & 255 & 255 \\ 0 & 0 & 0 & 255 & 255 & 255 & 255 \\ 0 & 0 & 0 & 0 & 255 & 255 & 255 \\ 0 & 0 & 0 & 0 & 0 & 255 & 255 \\ 0 & 0 & 0 & 0 & 0 & 0 & 255 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Erode Filter

Morphology Kernel =

255	255	0	0	0	255	255
255	0	0	0	0	0	255
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
255	0	0	0	0	0	255
255	255	0	0	0	255	255

Geometry

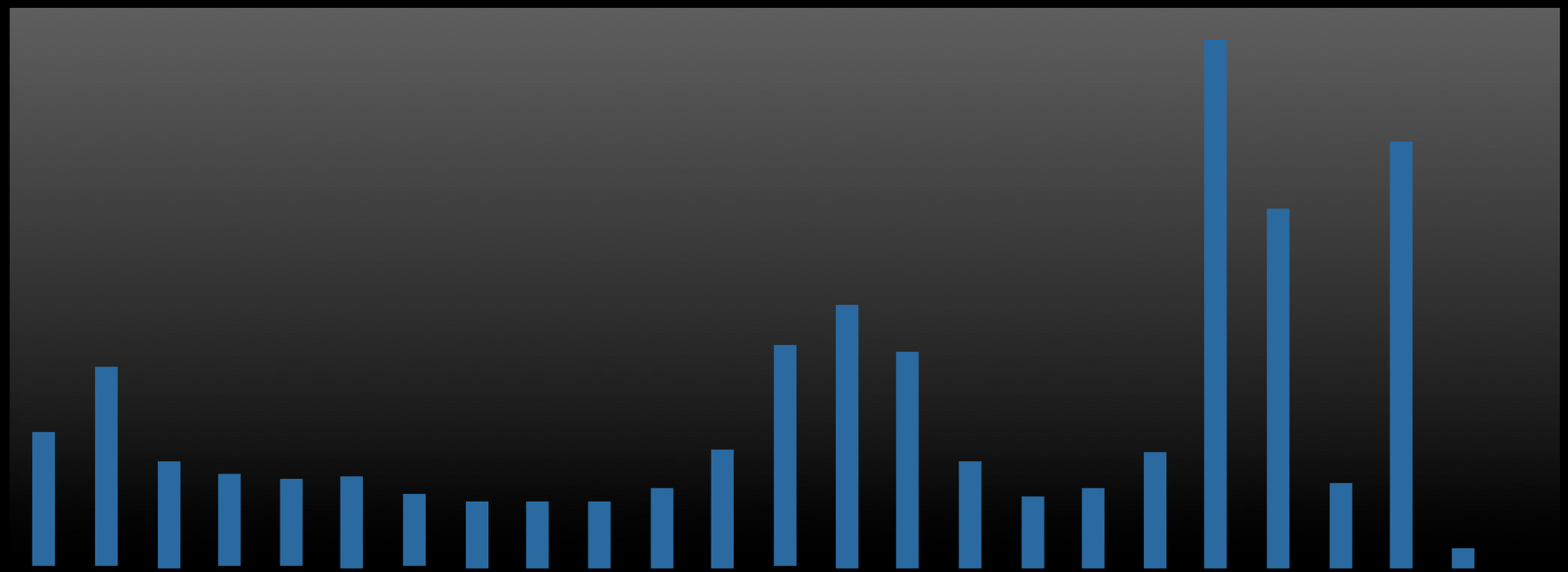
- Rotate
- Shear
- Translate
- Reflect
- Reduce/Enlarge
- Affine transform

Histogram

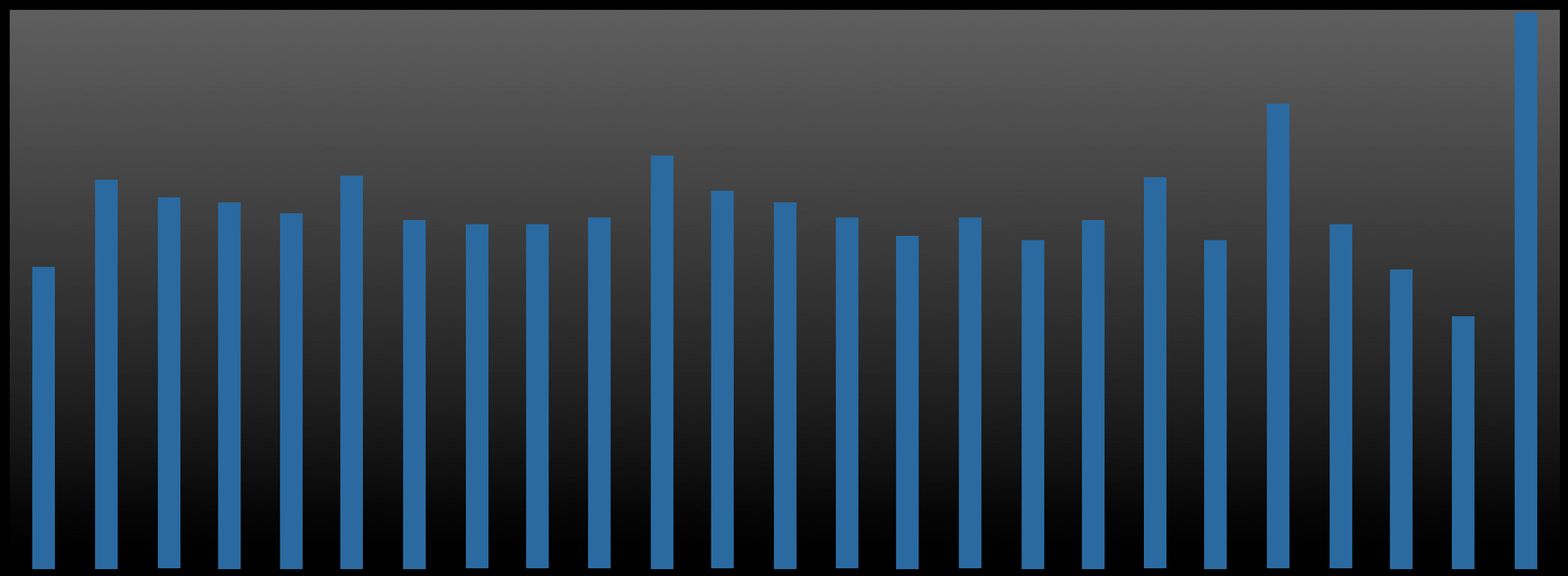
- Uses:
  - Color balance
  - Lookup tables
  - Range control
  - Contrast distribution

Intensity Distribution

Before



After



Number of pixels in each intensity range

Data Structures and API

Data Types and Layouts

- Data Types
  - 8 bit integer per color component (channel)
  - 32 bit floating point value per color component
- Data Layouts
  - Planar (each channel in its own array)
  - ARGB interleaved

Buffer 1
r r r r r r r r r r r r r r r r
r r r r r r

Buffer 2
g g g g g g g g g g g g g g g g
g g g g g g

Buffer 3
b b b b b b b b b b b b b b b b
b b b b b b

Single Buffer
a r g b a r g b a r g b
a r g b a r g b a r g b
a r g b a r g b a r g b
a r g b a r g b a r g b
a r g b a r g b a r g b

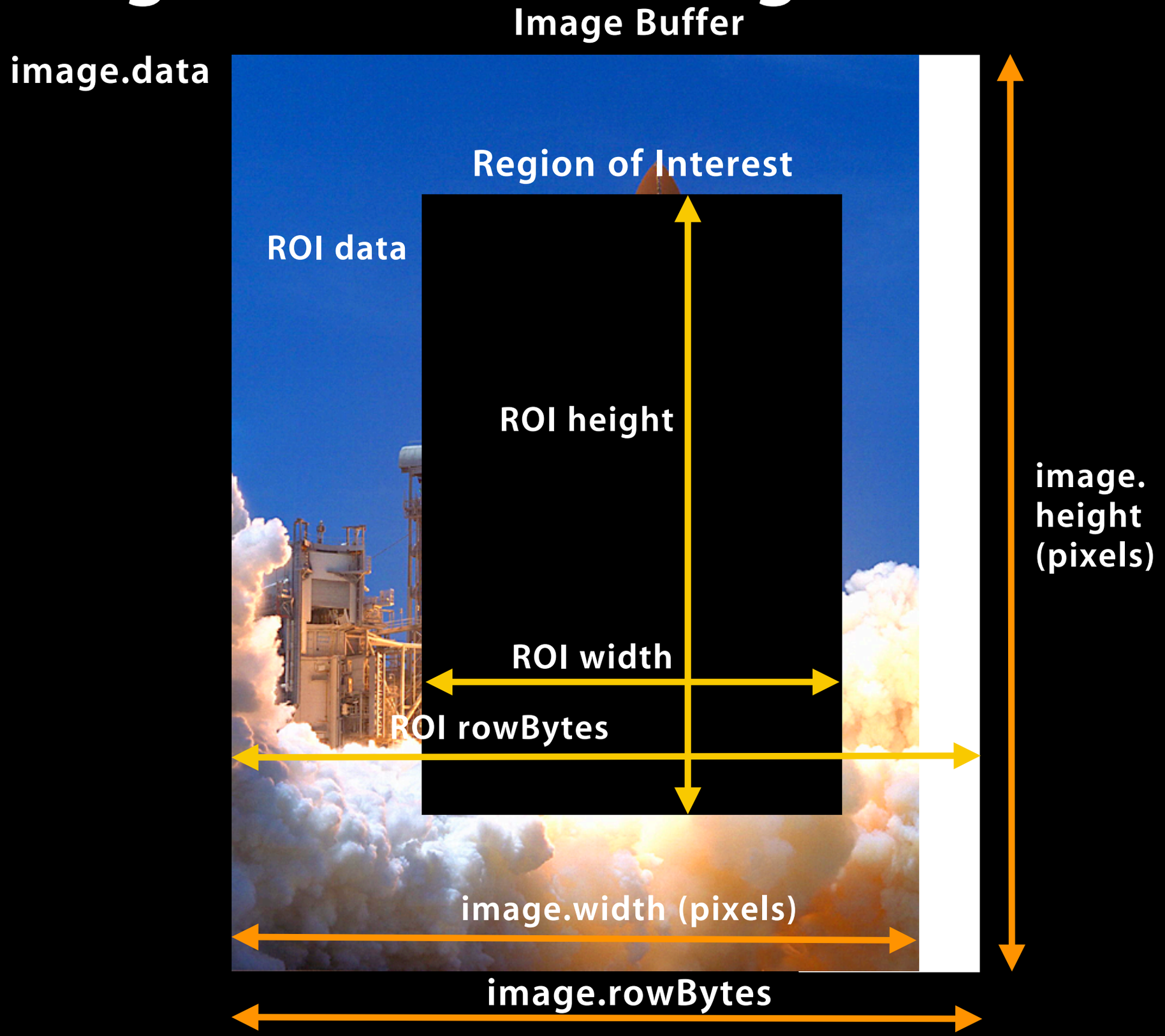
Data Conversion

- Planar integer to/from Planar floating point
- Planar integer to/from Interleaved integer
- Planar floating point to/from Interleaved floating point
- (RGBA handled by passing address of buffer–1)

Data Structures—vImage_Buffer

```
typedef struct vImage_Buffer
{
    void        *data;      /* Pointer to the top left pixel of the buffer.  */
    u_int32_t    height;    /* The height (in pixels) of the buffer          */
    u_int32_t    width;     /* The width (in pixels) of the buffer           */
    u_int32_t    rowBytes;  /* The number of bytes in a pixel row            */
} vImage_Buffer;
```

Image Buffers and Regions of Interest

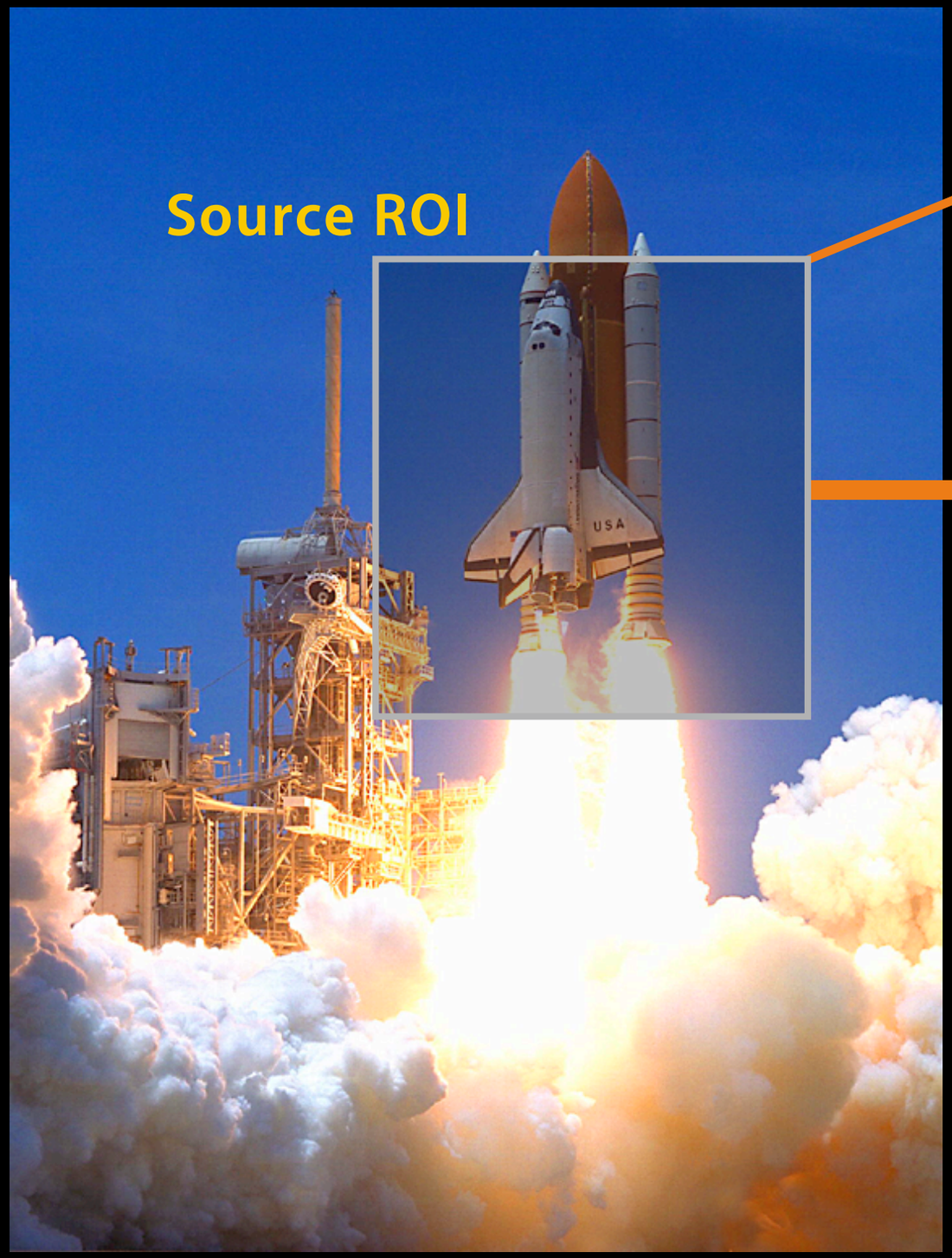


Function Call Example: Equalization

```
vImage_Error vImageEqualization_Planar8(
    const vImage_Buffer *src,          // source buffer
    const vImage_Buffer *dest,         // destination buffer
    vImage_Flags flags );              // flags word
```

Some Functions Require Both Image Buffers and ROIs

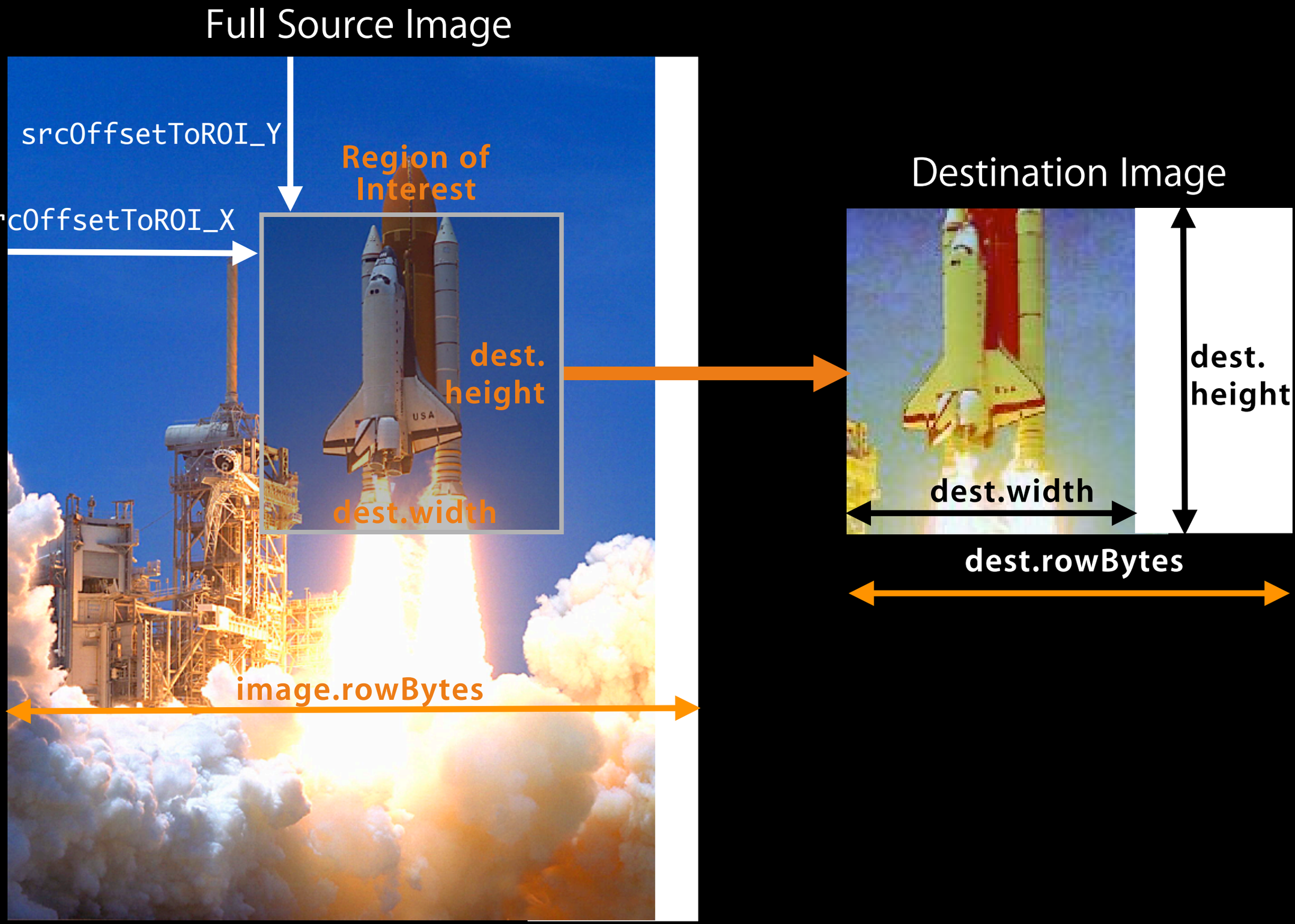
Source Image



Convolution Kernel

Destination Image

Image Buffers and Regions of Interest

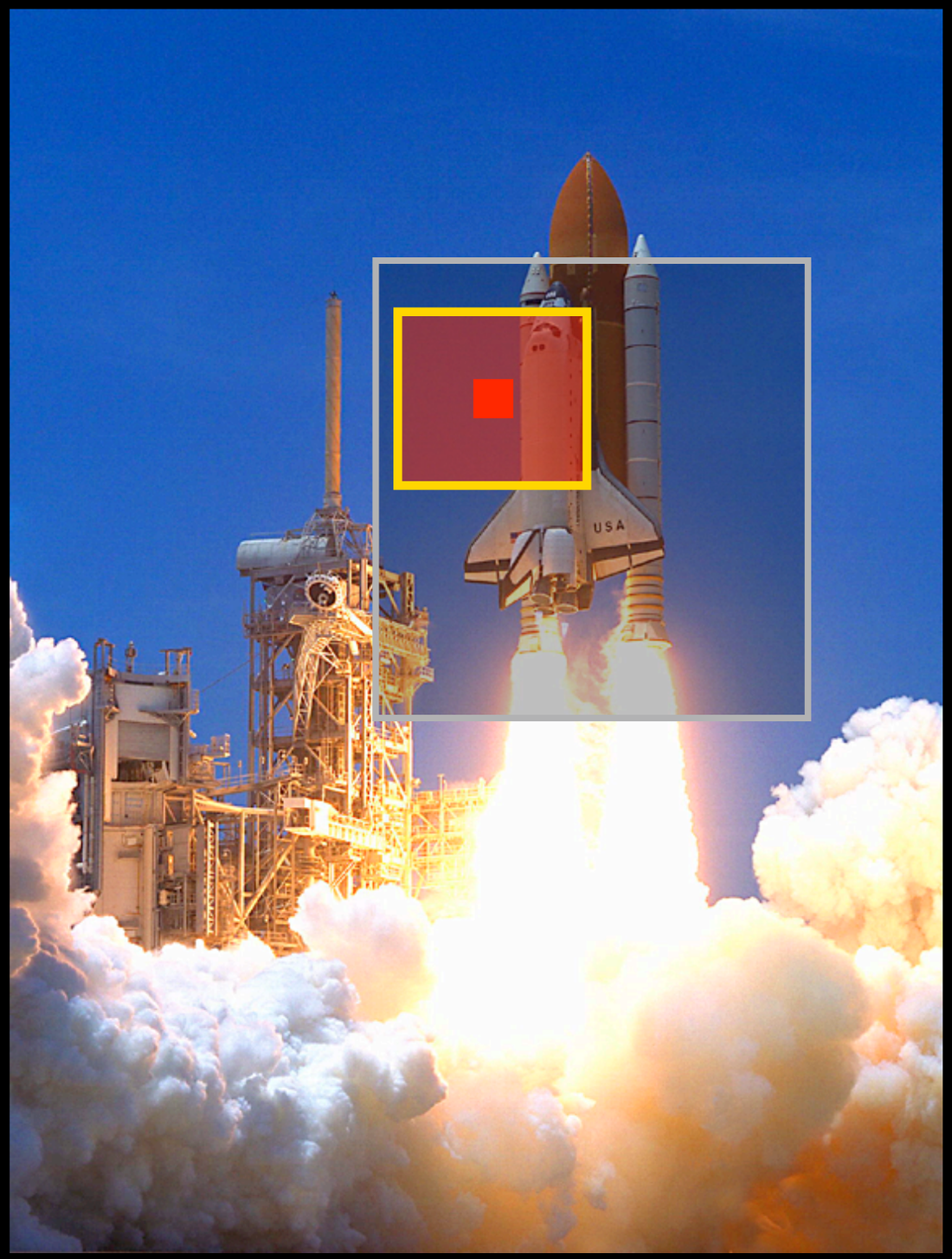


Function Call Example: Convolution

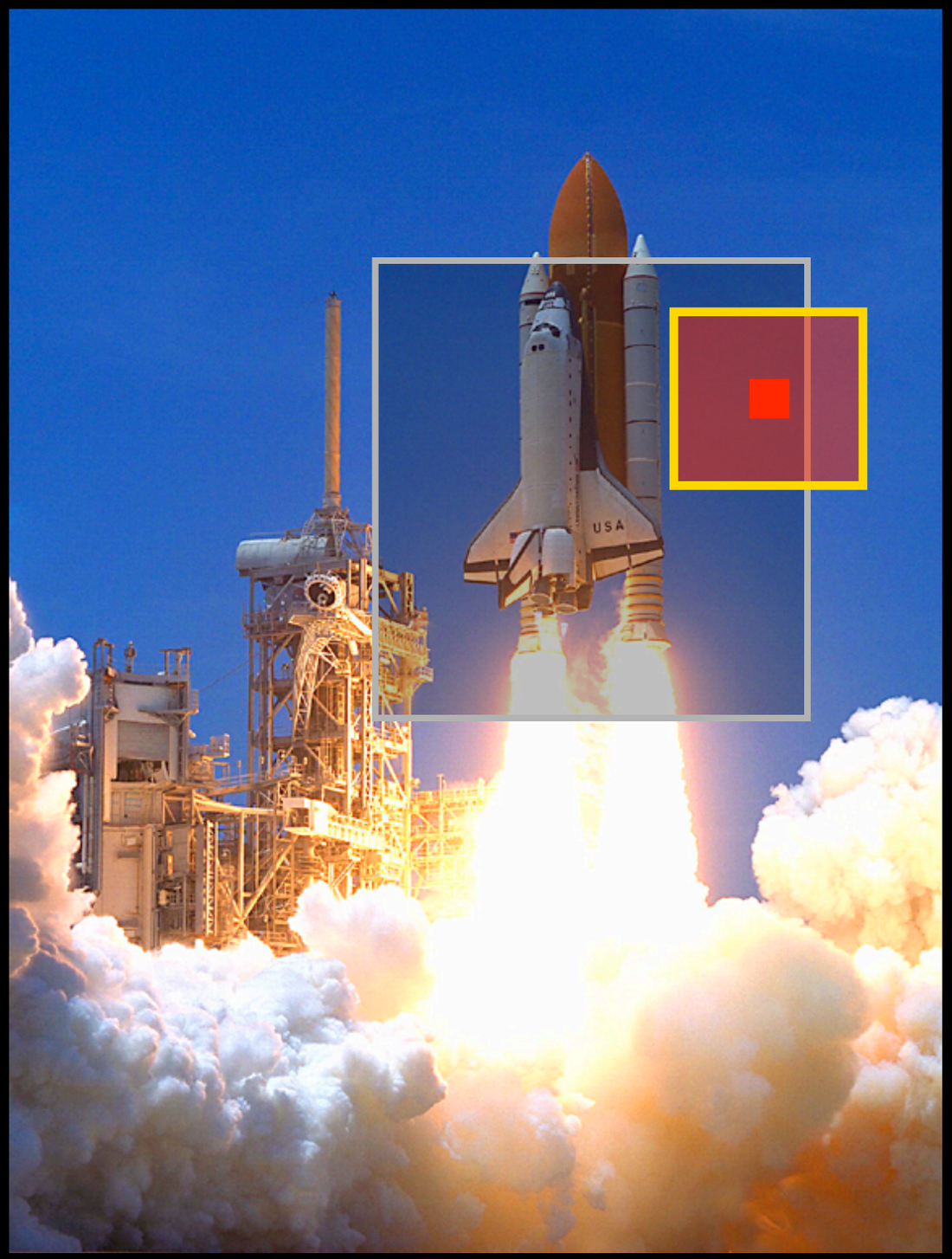
```
vImage_Error vImageConvolve_Planar8(
    const vImage_Buffer     *src,               // Source image buffer
    const vImage_Buffer     *dest,              // Destination image buffer
    unsigned int            srcOffsetToROI_X,   // Source image X-offset to ROI
    unsigned int            srcOffsetToROI_Y,   // Source image Y-offset to ROI
    short                   *kernel,            // Ptr to Convolution Kernel
    unsigned int            kernel_height,      // Height of convolution kernel
    unsigned int            kernel_width,       // Width of convolution kernel
    int                     divisor,            // Divisor for normalization
    Pixel_8                 backgroundColor,    // background color fill character
    vImage_Flags            flags );            // flags
```

Three Computational Cases

Matrix entirely inside ROI



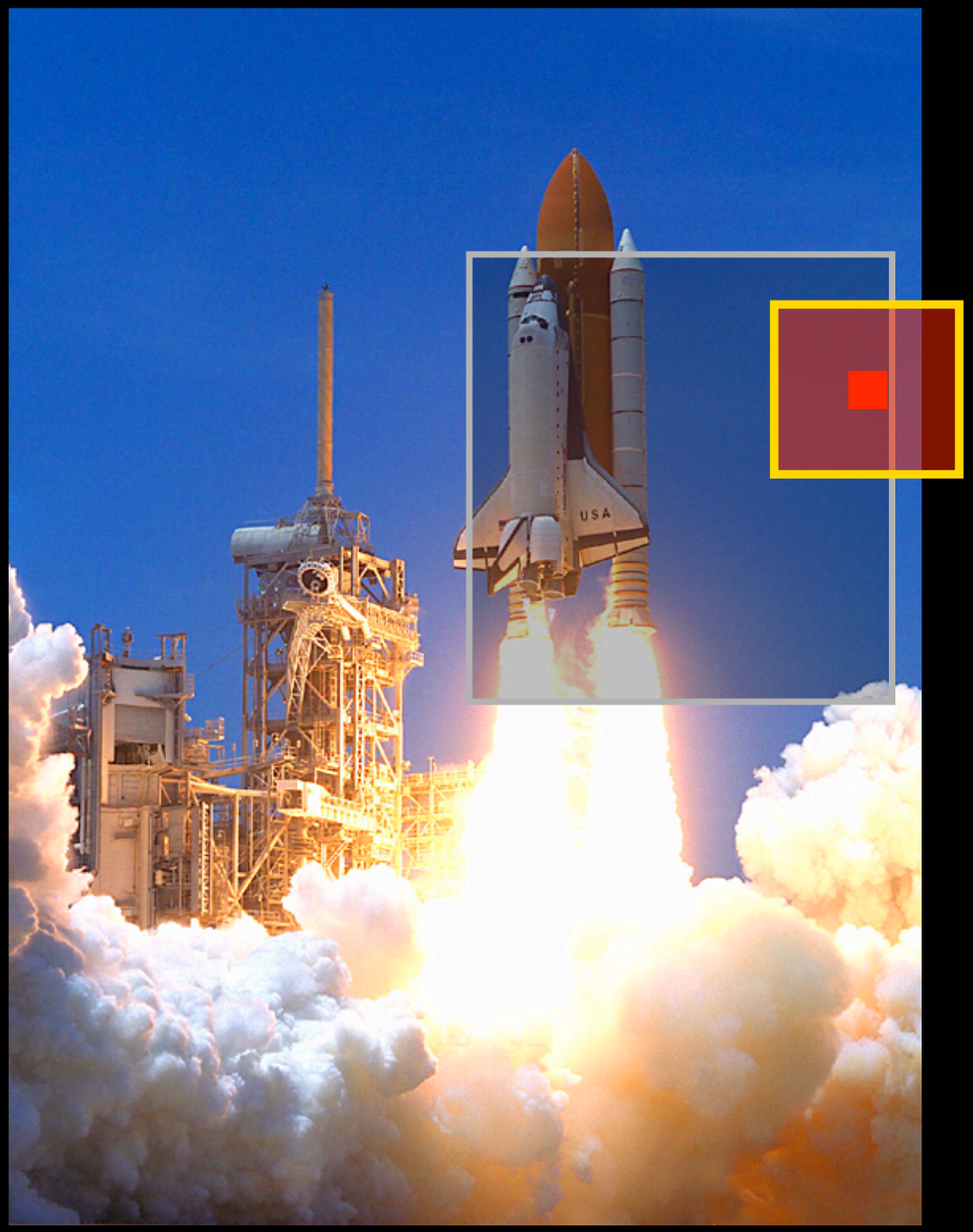
Matrix outside ROI but inside image



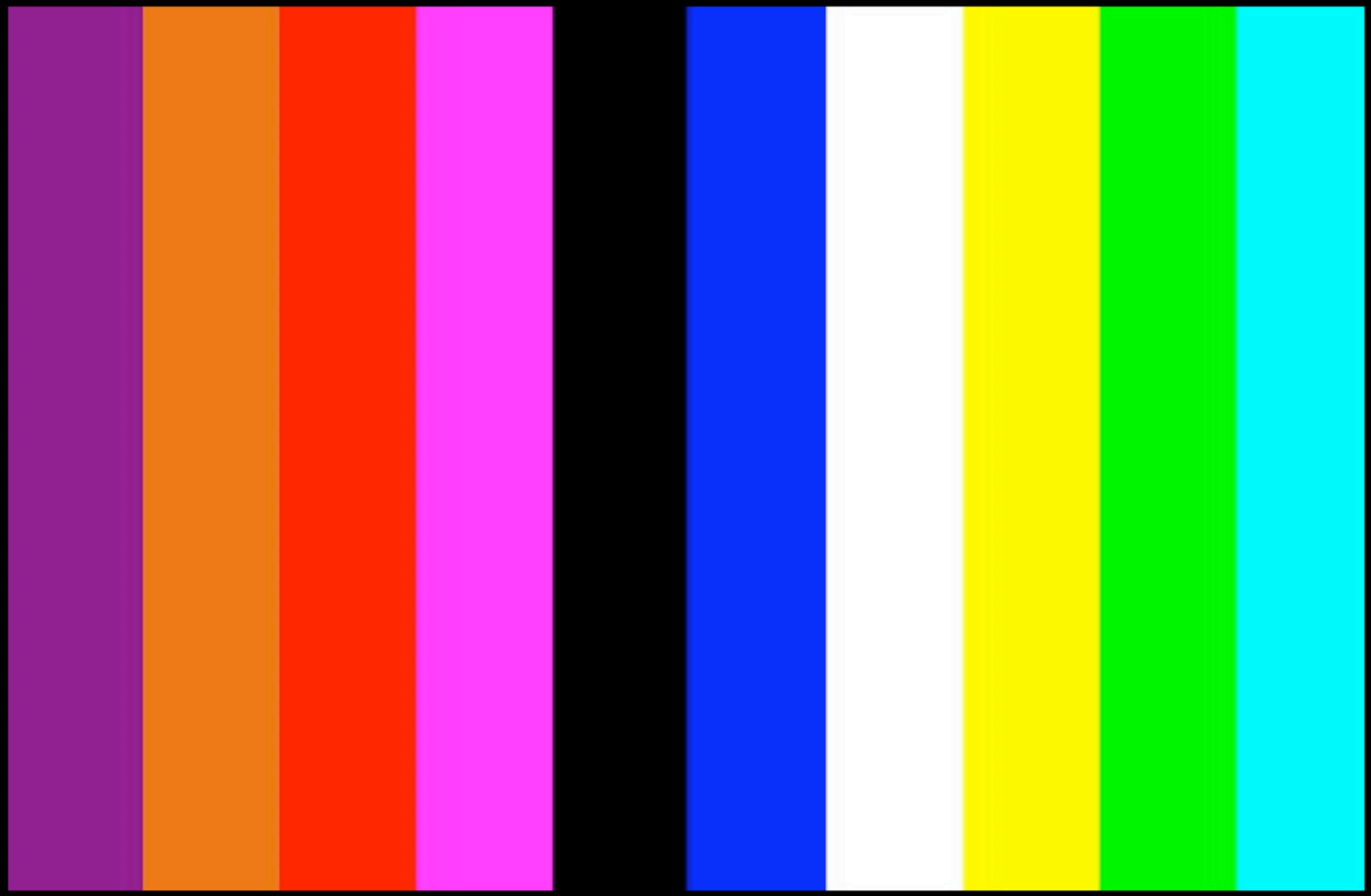
Matrix Extends Beyond Full Image

Choices for handling

- Background color
- Edge extend
- Copy-in-Place

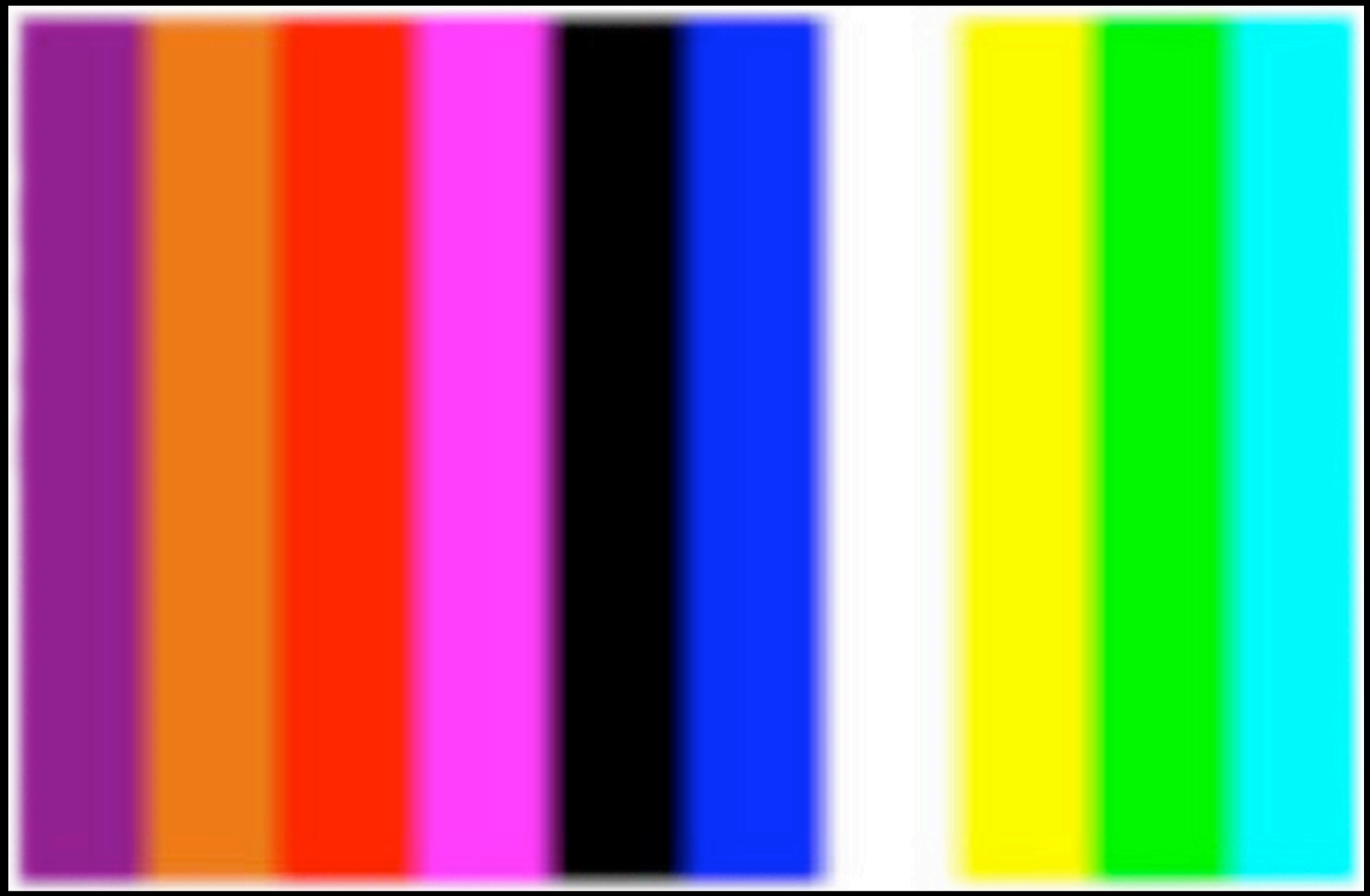


Original Image

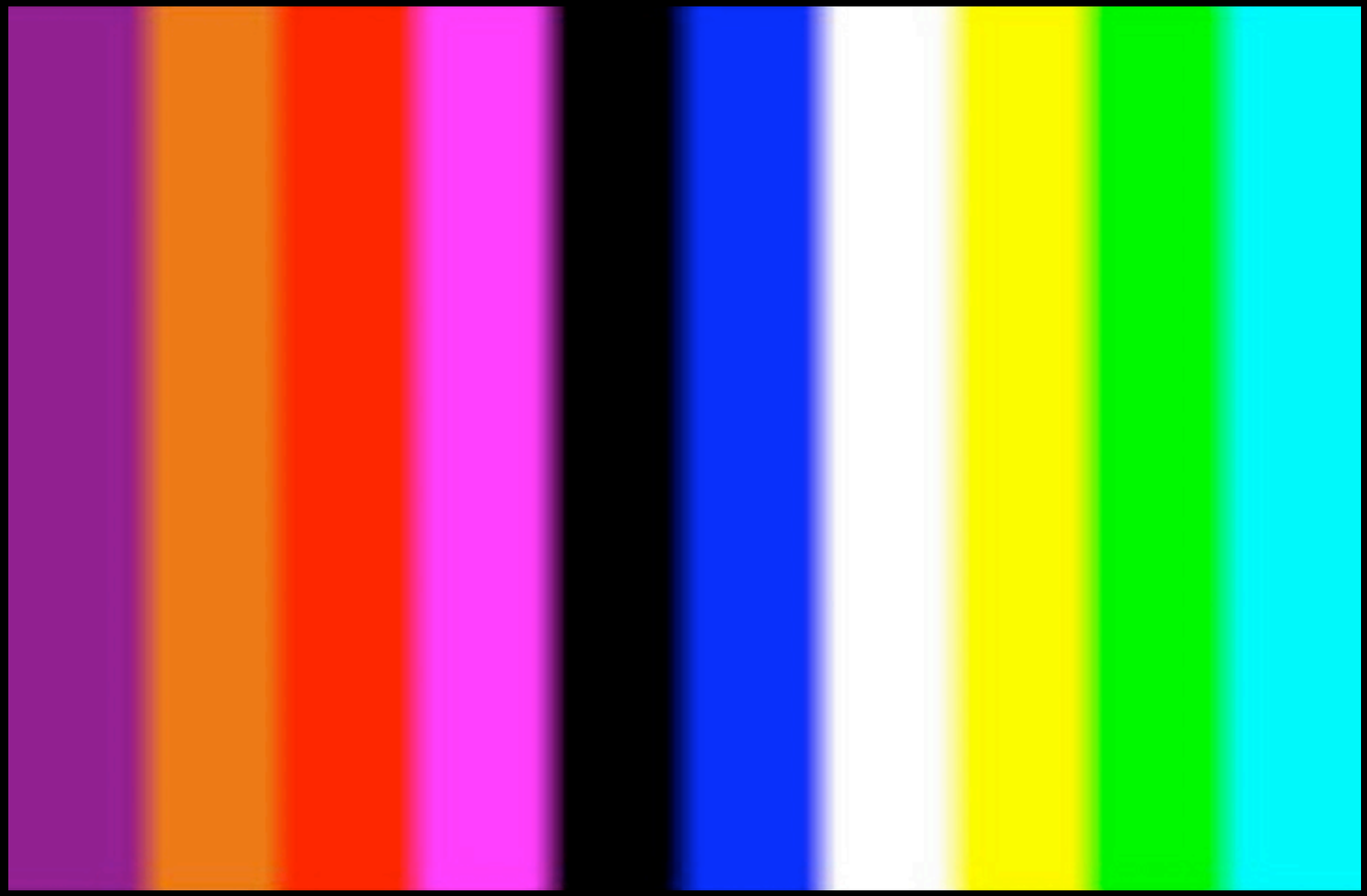


Background Color—Black

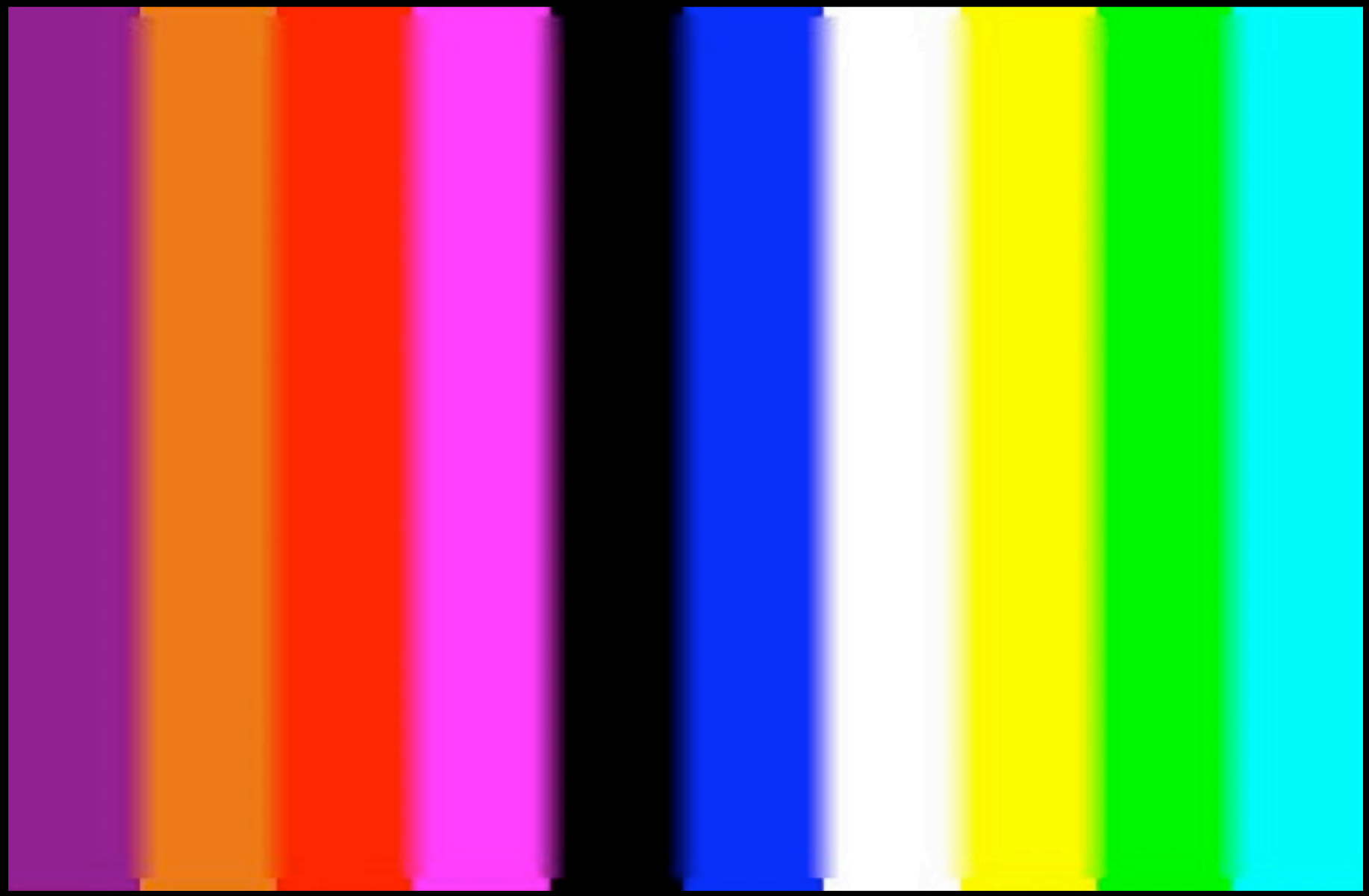
Background Color—White



Edge Extend



Copy-In-Place



Features

Transparently optimized for all Apple processors
Multi-Processor safe
Standard part of Panther
Unencapsulated Data structures
No license fees!

Implementation Techniques and Performance

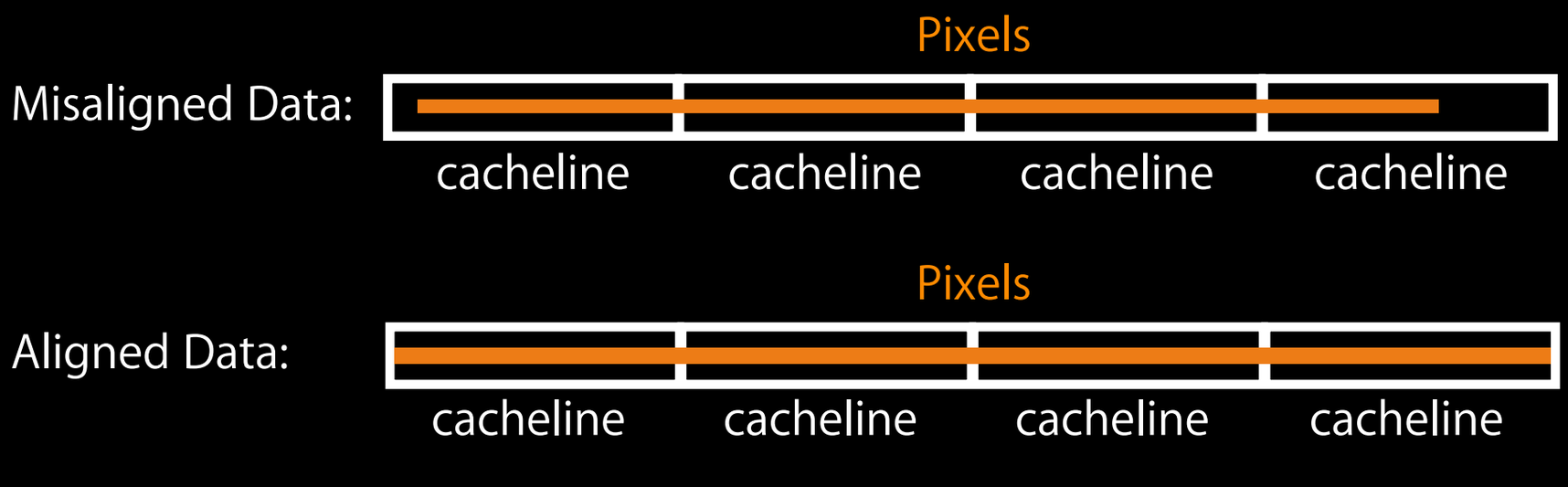
Ian Ollmann, Ph.D.
Sr. Software Engineer, Vector and Numerics Group

Performance Techniques for Getting the Most From vImage

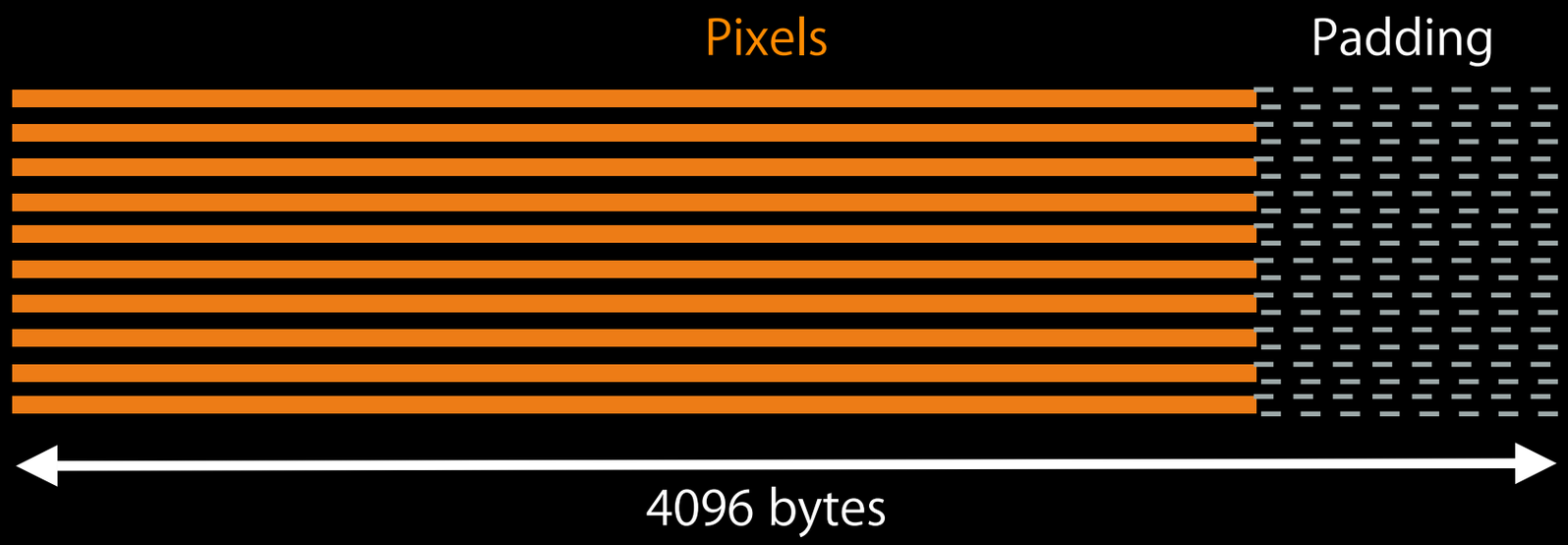
- Optimize alignment
- Consider cache usage/tiling
- MP/Real Time considerations

Alignment

- Align pixel rows to cacheline boundaries if possible



- Avoid rowbytes of integer power of 2

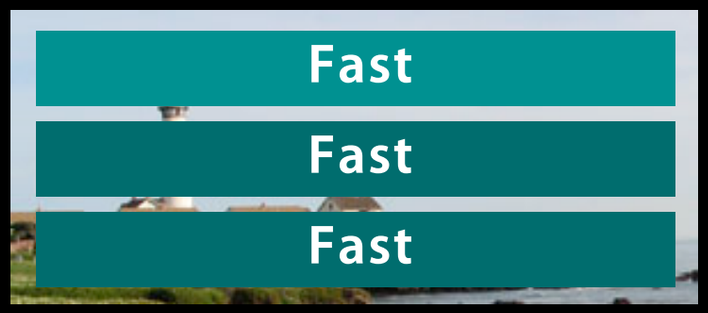


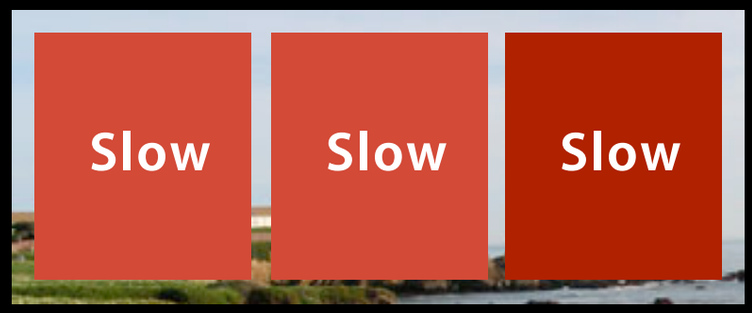
Tiling

- **Problem:** Many vImage functions are I/O-bound
- **Solution:** Organize for better cache reuse
- Tiling/strip mining

Tiling Tips

- Measure time/pixel for small and large images to see if tiling will help
  - Small: <256 kB uncompressed
  - Large: >3 MB uncompressed
- Size
  - Tiles usually are best at 16-32kB in size
  - Wide is better than tall/square

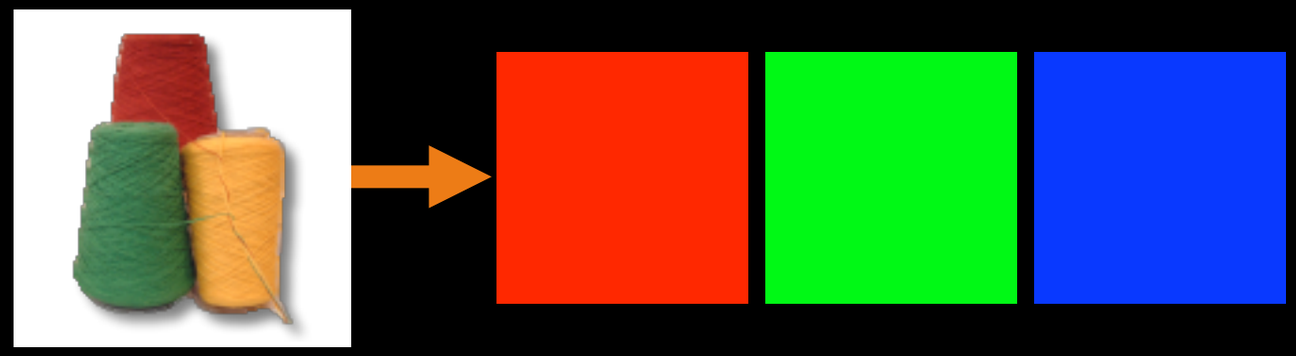




- Use `kvImageDoNotTile` to stop vImage internal tiling

Planar Data

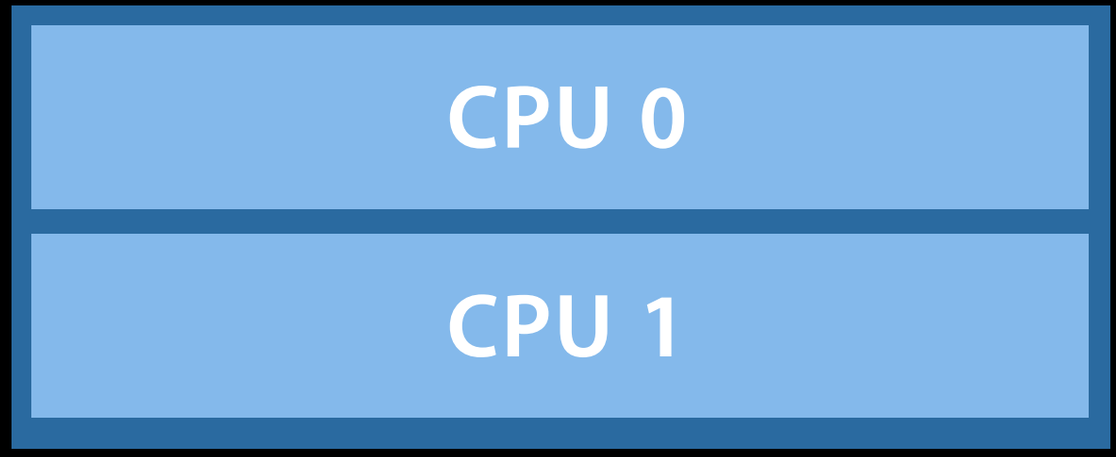
Almost always faster



- Saves work and cache space if you have data channels (e.g. alpha) that you don't use all the time
- Tiling is built-in
  - Operate on one channel at a time
  - Works nicely for Convolve/Morphology/Geometry
- Removes a permute/unpermute step
  - Most of our vector code converts to planar format in register, does the calculation, then converts back

Multiprocessing/Real-time

- All functions in vImage are single threaded
- Non-blocking with optional temp buffers
- Better speed through tiling
  - Separate the tiles vertically:



  - Careful about kernel using functions and race conditions

Optimization Techniques We Used

- Data-rich programming approach
- Experimentally driven optimization
- Tiling
- Processor specific tuning
- Velocity Engine!

Data Rich Programming

- Current G4s are very parallel machines
- GPUL is more so
- Each cycle on G4, up to:
  - 4 integer ops
  - 10 scalar floating point ops
  - 40 vector floating point ops
  - 128 vector integer ops
  - Can be in process
- We structure our code so that we have at least this much parallelism to keep serial data dependencies from stalling the pipelines

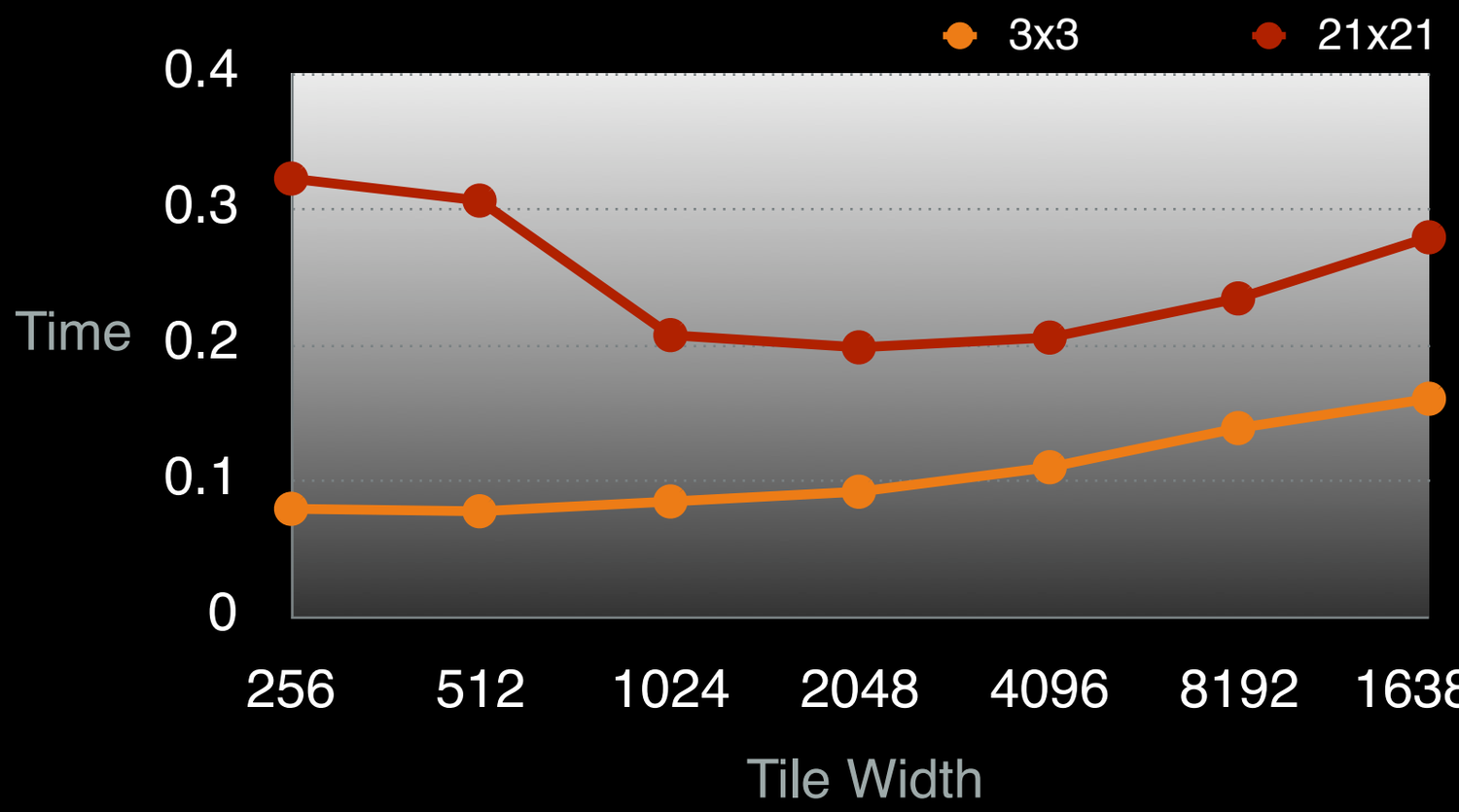
Data Rich Programming in Practice

Key: Never do one thing at a time

- Common approaches
- Unroll loops
- Identify and eliminate compiler aliasing
- Eliminate LSU bottlenecks
- Hide long latencies with other work
- Eliminate branching
- Keep the execution units busy at all times
- prefetch data so it is there when you need it

Tiling

- Some functions (esp. multipass algorithms) do their own tiling



- Hand tuned for best performance
  - We tried a series of different tile sizes and shapes
  - Very wide, short rectangles usually the best
  - Workhorse leaf functions are tuned per processor

Velocity Engine

- vImage uses the Velocity Engine across the board
- All segments (except Histogram) will be or are vectorized for Panther for all data types
- Typical speedups over scalar code 4-10x
- Hand optimized scalar code for G3
- In some cases (Resampling) we use the speed to bring you better image quality

This is a beta release!

Experimentally Driven Optimization

- When optimizing never guess about performance
- Profile always
  - gprof
  - Sampler
  - CHUD / Shark
  - CPU simulators like SimG4
- Inspect compiler output—there are surprises!

CHUD Tools

Computer Hardware Understanding Developer Tools

Eric Miller

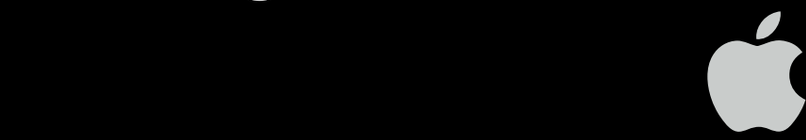
Sr Scientist, Architecture and Performance Group

Introduction to CHUD Tools

- Suite of performance analysis tools
- Low-level access to performance monitors
  - Processor
  - Memory controller
  - Operating system
- Helps you find problems and improve your code
- Freely available on the web and the developer tools CD!

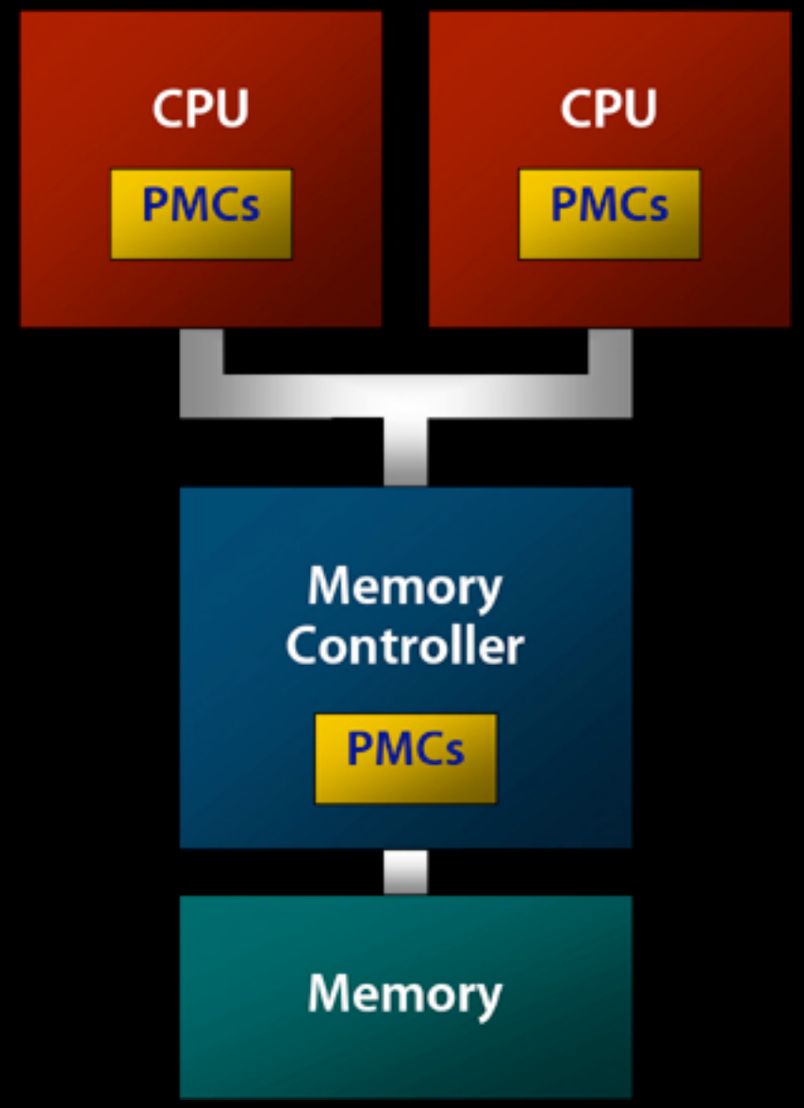
CHUD Tools 3.0

- Profiling
  - Shark—instruction-level profiler
  - MONster—spreadsheet for performance events
  - Saturn—function call graph visualizer
- Tracing
  - Amber—instruction tracer
  - Acid—trace analyzer
  - SimG4 and SimG5—cycle-accurate CPU simulators
- Instrumentation
  - CHUD.framework—API for performance tuning



Performance Counters

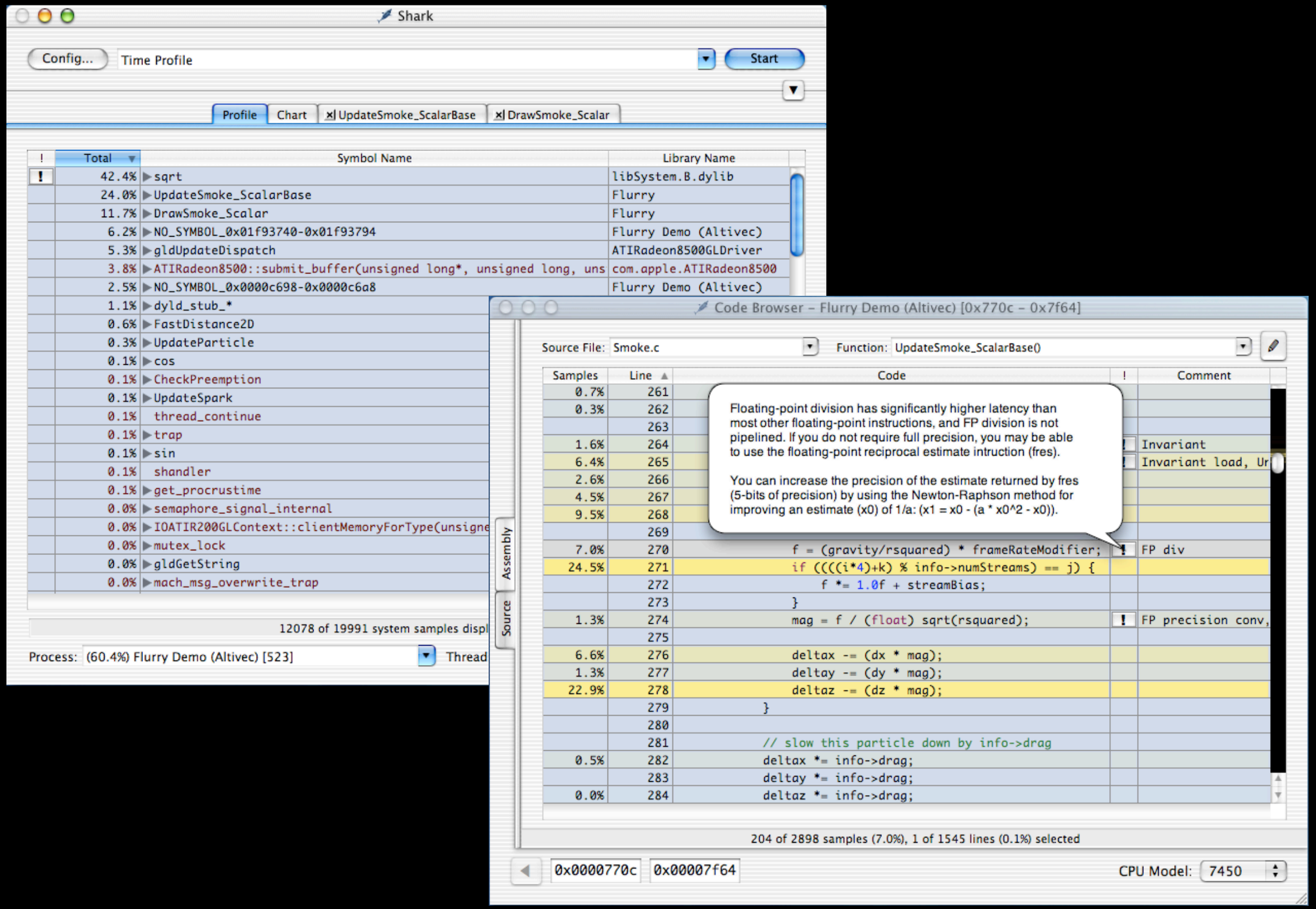
- Apple systems have performance monitor counters (PMCs)
- PMCs record performance "events"
  - Cache misses
  - Instructions completed
  - Execution stall cycles
  - Page faults
- Use CHUD Tools to configure, control and view the PMCs



Shark

- Profile a process, thread or entire system
  - Time profile—visualize performance hot spots
  - Event profile—relate hardware events to source
  - Capture everything—drivers, kernel, applications
  - Very low overhead
- Automated Analysis
  - Annotated source code and disassembly
  - Static analysis to find sub-optimal code
  - Optimization tips
- Save and review sessions
- Command line version

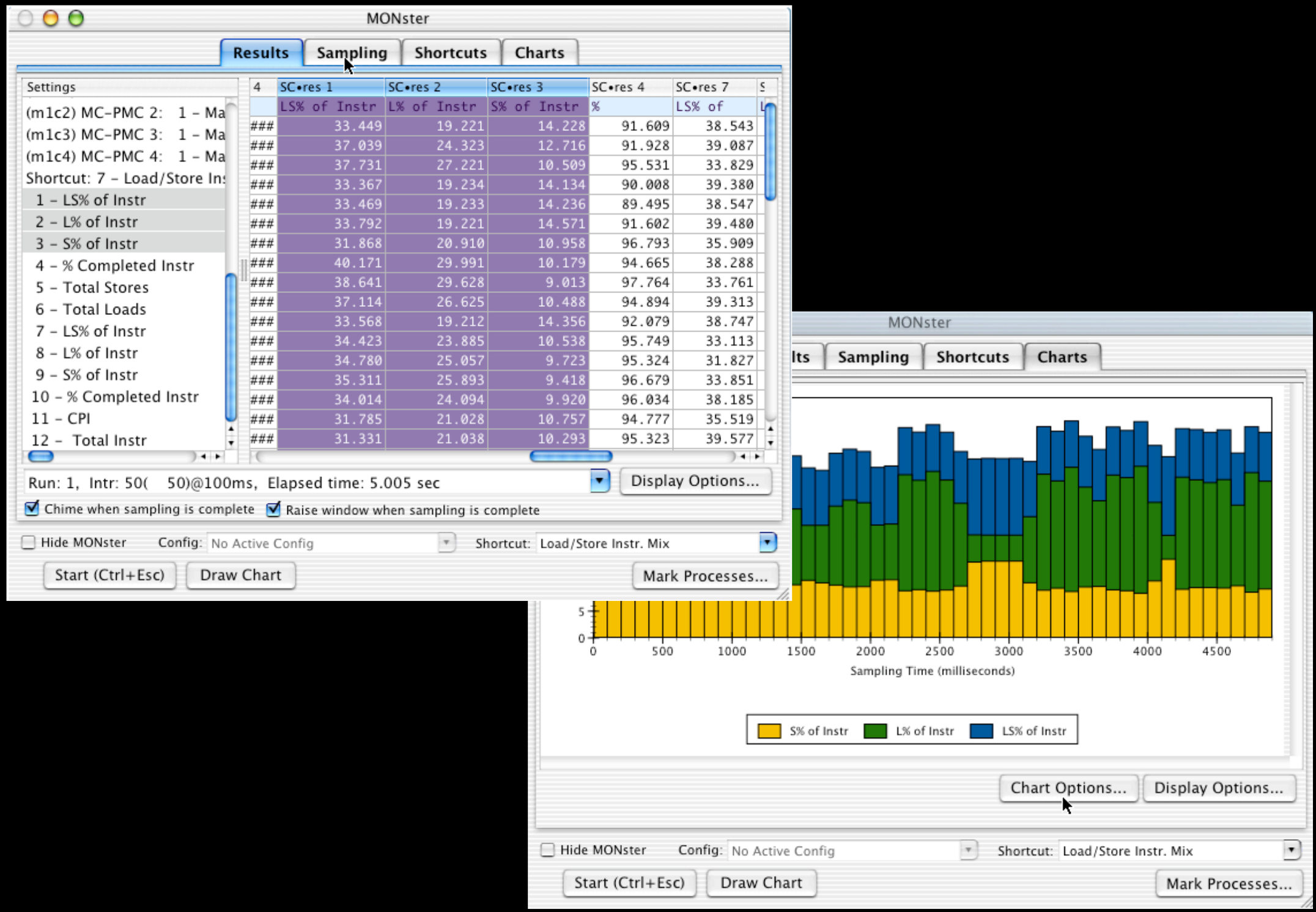
Shark


Shark
Config...
Time Profile
Start
Profile
Chart
UpdateSmoke_ScalarBase
DrawSmoke_Scalar
! Total Symbol Name Library Name
! 42.4% sqrt libSystem.B.dylib
24.0% UpdateSmoke_ScalarBase Flurry
11.7% DrawSmoke_Scalar Flurry
6.2% NO_SYMBOL_0x01f93740-0x01f93794 Flurry Demo (Altivec)
5.3% gldUpdateDispatch ATIRadeon8500GLDriver
3.8% ATIRadeon8500::submit_buffer(unsigned long*, unsigned long, uns com.apple.ATIRadeon8500
2.5% NO_SYMBOL_0x0000c698-0x0000c6a8 Flurry Demo (Altivec)
1.1% dyld_stub_*
0.6% FastDistance2D
0.3% UpdateParticle
0.1% cos
0.1% CheckPreemption
0.1% UpdateSpark
0.1% thread_continue
0.1% trap
0.1% sin
0.1% shandler
0.1% get_procrustime
0.0% semaphore_signal_internal
0.0% IOATIR200GLContext::clientMemoryForType(unsigne
0.0% mutex_lock
0.0% gldGetString
0.0% mach_msg_overwrite_trap
12078 of 19991 system samples displ
Process: (60.4%) Flurry Demo (Altivec) [523]
Thread
Code Browser – Flurry Demo (Altivec) [0x770c – 0x7f64]
Source File: Smoke.c
Function: UpdateSmoke_ScalarBase()
Samples Line Code ! Comment
0.7% 261
0.3% 262
263
1.6% 264 ! Invariant
6.4% 265 ! Invariant load, Ur
2.6% 266
4.5% 267
9.5% 268
269
7.0% 270 f = (gravity/rsquared) * frameRateModifier; ! FP div
24.5% 271 if ((((i*4)+k) % info->numStreams) == j) {
272 f *= 1.0f + streamBias;
273 }
1.3% 274 mag = f / (float) sqrt(rsquared); ! FP precision conv,
275
6.6% 276 deltax -= (dx * mag);
1.3% 277 deltay -= (dy * mag);
22.9% 278 deltaz -= (dz * mag);
279 }
280
281 // slow this particle down by info->drag
0.5% 282 deltax *= info->drag;
283 deltay *= info->drag;
0.0% 284 deltaz *= info->drag;
Floating-point division has significantly higher latency than most other floating-point instructions, and FP division is not pipelined. If you do not require full precision, you may be able to use the floating-point reciprocal estimate intruction (fres).
You can increase the precision of the estimate returned by fres (5-bits of precision) by using the Newton-Raphson method for improving an estimate (x0) of 1/a: (x1 = x0 - (a * x0^2 - x0)).
Assembly
Source
204 of 2898 samples (7.0%), 1 of 1545 lines (0.1%) selected
0x0000770c
0x00007f64
CPU Model: 7450

MONster

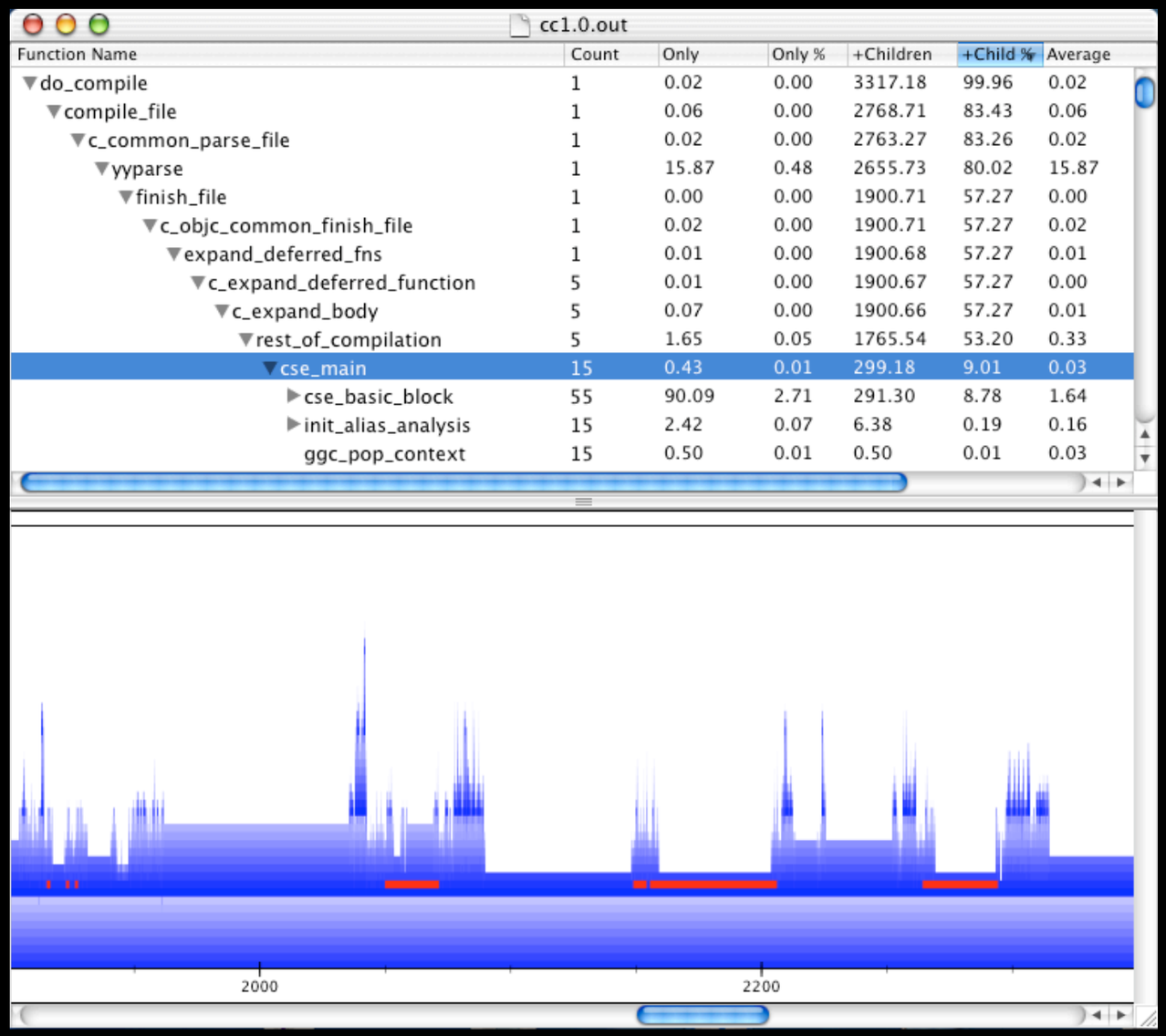
- Configure Performance Monitor Counters (PMCs)
- Collect PMC data based on:
  - Timed intervals, event counts or hotkey
- Shortcuts—compute performance metrics:
  - Memory bandwidth, cycles per instruction
  - Define your own with infix expressions
- View results in tabular or chart form
- Save and review sessions
- Command line version

MONster

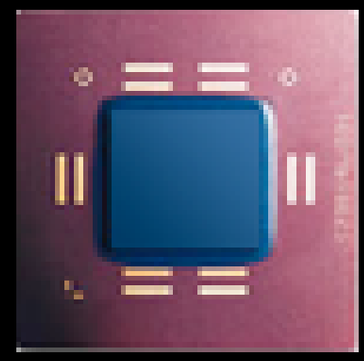

MONster
Results
Sampling
Shortcuts
Charts
Settings
(m1c2) MC-PMC 2: 1 - Ma
(m1c3) MC-PMC 3: 1 - Ma
(m1c4) MC-PMC 4: 1 - Ma
Shortcut: 7 - Load/Store In
1 - LS% of Instr
2 - L% of Instr
3 - S% of Instr
4 - % Completed Instr
5 - Total Stores
6 - Total Loads
7 - LS% of Instr
8 - L% of Instr
9 - S% of Instr
10 - % Completed Instr
11 - CPI
12 - Total Instr
4 | SC•res 1 | SC•res 2 | SC•res 3 | SC•res 4 | SC•res 7
LS% of Instr | L% of Instr | S% of Instr | % | LS% of
| 33.449 | 19.221 | 14.228 | 91.609 | 38.543
| 37.039 | 24.323 | 12.716 | 91.928 | 39.087
| 37.731 | 27.221 | 10.509 | 95.531 | 33.829
| 33.367 | 19.234 | 14.134 | 90.008 | 39.380
| 33.469 | 19.233 | 14.236 | 89.495 | 38.547
| 33.792 | 19.221 | 14.571 | 91.602 | 39.480
| 31.868 | 20.910 | 10.958 | 96.793 | 35.909
| 40.171 | 29.991 | 10.179 | 94.665 | 38.288
| 38.641 | 29.628 | 9.013 | 97.764 | 33.761
| 37.114 | 26.625 | 10.488 | 94.894 | 39.313
| 33.568 | 19.212 | 14.356 | 92.079 | 38.747
| 34.423 | 23.885 | 10.538 | 95.749 | 33.113
| 34.780 | 25.057 | 9.723 | 95.324 | 31.827
| 35.311 | 25.893 | 9.418 | 96.679 | 33.851
| 34.014 | 24.094 | 9.920 | 96.034 | 38.185
| 31.785 | 21.028 | 10.757 | 94.777 | 35.519
| 31.331 | 21.038 | 10.293 | 95.323 | 39.577
Run: 1, Intr: 50(50)@100ms, Elapsed time: 5.005 sec
Display Options...
Chime when sampling is complete
Raise window when sampling is complete
Hide MONster
Config: No Active Config
Shortcut: Load/Store Instr. Mix
Start (Ctrl+Esc)
Draw Chart
Mark Processes...
MONster
lts
Sampling
Shortcuts
Charts
5
0
0
500
1000
1500
2000
2500
3000
3500
4000
4500
Sampling Time (milliseconds)
S% of Instr
L% of Instr
LS% of Instr
Chart Options...
Display Options...
Hide MONster
Config: No Active Config
Shortcut: Load/Store Instr. Mix
Start (Ctrl+Esc)
Draw Chart
Mark Processes...

Saturn

- Record function call history
- Visualize call tree
- Instruments functions at entry/exit with GCC
- For each function:
  - Call counts
  - PMC event counts
  - Execution times



Instruction Tracing and Simulation



- Amber—instruction tracer
- Acid—trace analyzer
  - Trace statistics
  - Memory footprint (gnuplot output)
  - Points out problematic instruction sequences
    - Dependency stalls, spilled loads, unaligned operands
- SimG4–PPC7400 cycle-accurate simulator
- SimG5–PPC970 cycle-accurate simulator (coming soon)

CHUD.framework

- Instrument your source code
  - Start and stop MONster or Shark
  - Directly read and report on PMCs
- Write your own performance analysis tools
  - Control Performance Monitor Counters
  - Collect information about system hardware
    - CPU, memory controller, bus
- Extensive HTML reference guide

CHUD.framework Example #1

```
#include <CHUD/chud.h>
…
chudInitialize();

// MONster or Shark is in remote mode
chudAcquireRemoteAccess();

chudStartRemotePerfMonitor(“run_01”);

your_important_function();

chudStopRemotePerfMonitor();

chudReleaseRemoteAccess();
```

CHUD.framework Example #2

```
chudInitialize();
chudAcquireSamplingFacility();

chudSetMarkFilter(chudAllCPUDev, chudFmodeOnlyMarked);
chudMarkPID(getpid(), TRUE);


chudSetCounterEvent(chudAllCPUDev, PMC_1, 1);
chudSetCounterEvent(chudAllCPUDev, PMC_2, 2);

chudClearCounters();
chudStartCounters();

important_function();

chudStopCounters();

do_something_with_results();
chudReleaseSamplingFacility();
```

For More Information

- On the Web:
  http://developer.apple.com/tools/debuggers.html
- Send Raves, Rants, and Requests:
  chud-tools-feedback@group.apple.com

Wrap Up

Roadmap

506 CHUD Performance Optimization Collect Data and Measure Performance	**Marina** **Wed, 2:00pm**
305 Tuning Software With Performance Tools	**Presidio** **Wed, 5:00pm**
507 Mac OS X High Performance Math Libraries	**Marina** **Thu, 9:00am**
416 Carbon: Optimizing Performance	**Presidio** **Thu, 2:00pm**

Who to Contact

Ali Sazegari, Ph.D.
Manager, Vector and Numerics Group
sazegari@apple.com

Ian Ollmann, Ph.D.
Sr. Software Engineer
iano@apple.com

Robert Murley
Sr. Engineering Scientist
murley@apple.com

Paul Chang
Software Engineer
pchang@apple.com

http://developer.apple.com/wwdc2003/urls.html

Reference Library

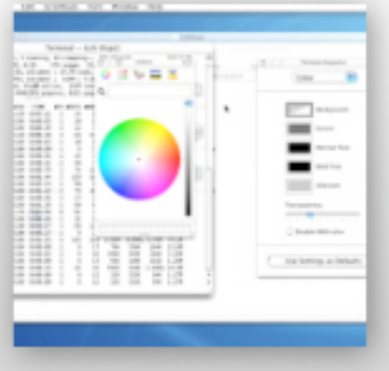
Documentation

Accelerate
vecLib
http://developer.apple.com/hardware/ve/vector_libraries.html

vImage
http://developer.apple.com/?.html

Velocity Engine
http://developer.apple.com/hardware/ve/index.html

Q&A



Ali Sazegari, Ph.D.
Manager, Vector and Numerics Group
sazegari@apple.com

http://developer.apple.com/wwdc2003/urls.html

WWDC 2003

Worldwide Developers Conference